A Practical Guide to Developing and Using Mesh and Geometry Frameworks for Advanced Meshing and Computational Software

Rao Garimella (rao@lanl.gov)
T-5, Theoretical Division
Los Alamos National Laboratory,
Los Alamos, NM

LA-UR-11-11693

Abstract

The representation and management of meshes is one of the most central and critical parts of advanced application software solving systems of PDEs. This is particularly true for applications that solve the PDEs for complex geometric domains using general unstructured meshes distributed across large numbers of processors. In recent years, several mesh frameworks have been developed to help applications represent and manage meshes effectively without having to develop this functionality themselves. This allows application developers to focus their energy on their areas of expertise rather than designing and maintaining code for handling mesh data.

In this short course, participants will be introduced to basics of building advanced computational software using widely available mesh infrastructure libraries. Topics that will be covered will include:

- Components of advanced computational software using parallel, unstructured meshes
- Introduction to unstructured mesh representations in parallel environments
- Accessing and manipulating mesh data through APIs
- Description of some popular mesh infrastructure libraries
- Practical code examples using mesh infrastructure libraries
- The link between meshes and geometric models
- Frameworks for accessing geometric model data
- Handling analysis attributes like boundary conditions and material properties
- Managing field data on meshes
- Preprocessing for solver libraries
- Parallel I/O

Overview

- Introduction
- Application requirements
- Definitions
- Mesh data structure design
- Meshes and geometric models
- Parallel mesh management
- Mesh frameworks
- Example: Pseudo-shock propagation
- Geometric modeler interfaces
- Conclusions

Introduction

- **Large number of numerical methods for solving PDEs involve meshes**
- **Meshes (grids) are a discretized representation of a continuous domain**
- **Meshes may be structured (regular connectivity) or unstructured (irregular connectivity)**
- **Here we will only discuss unstructured meshes**
- **Simulations heading towards billion element meshes distributed over hundreds of thousands of processors**

Mesh Operations in Applications

Large scale applications typically require the following mesh operations:

- **Importing the mesh**
- **Representing the mesh**
- **Querying the topology and geometry of the mesh**
- **Modifying the mesh**
- **Querying and modifying mesh based data**
- **Exporting the mesh**

All these operations must account for mesh being distributed over a large number of processors

Mesh Design Questions

- How should the mesh be represented?
  - only elements and nodes? or edges/faces as well?
- What element types should the mesh admit?
  - tetrahedra, hexahedra or general polyhedra?
- What data structures should be used to represent mesh connectivity?
  - Arrays? Linked Lists?
- How should data be associated with mesh entities?
  - with the mesh entities themselves?
  - through coexisting array?
- What formats should be supported for reading and writing the mesh?
- And so on....

Typical Initial Design

- **Only elements and nodes supported**
  - “who needs faces?”
- **One or two element types are supported**
  - “no one knows the formulation for pyramids anyway”
- **Only downward connectivity, stored in linear arrays**
  - “upward connectivity is messy; just do a clever search”
- **Meshes are assumed to span only one processor**
  - “with 64 bit machines and super-light data structures, the mesh will always fit on one processor”
  - “we’ll never go beyond a million elements”
- **Arrays are accessed throughout the code**
  - “look at my clever trick to reinterpret the connectivity array”

A Year Later...

- **Application needs grow**
  - “we really need higher order elements”
  - “we need to store fluxes through faces”
- **More element types (pyramids, prisms or general polyhedra) needed**
  - “who thought they’d be able to generalize the math?”
- **Parallelization of code requested**
  - “we really need more resolution”
  - “we’ve added so many other arrays that our meshes don’t fit on one processor anymore”
- **Multiple meshes are required**
  - “want to couple multiple physics using different meshes”

Major code rewrite or adhoc patches affecting tens of thousands of lines of code

What is a Mesh Framework?

A software library that allows a user to represent, query and manipulate meshes through an application programming interface without having to manage mesh data structures directly.

Mesh Frameworks are also referred to as *mesh libraries*, *mesh infrastructure*, *mesh databases*.

Why Use Mesh Frameworks?

A mesh framework allows developers of mesh based software to go from code that looks like this:

```
nnodes = ielnnod[ie];
for (j = 0; j < nnodes; j++) {
  for (k = 0; k < ndims; k++) {
    coords[j][k] = pxyz[ielnode[ielem_offst[i]+j]]][k];
  }
}
```

to go to code that looks like this:

```
Elem_Get_Coords(elem(i),&nnodes,coords);
```

Specific Advantages of Using Mesh Frameworks

- **Applications do not worry about specifics of mesh data structures**
- **Applications access mesh data through well defined interfaces**
- **Operations such as parallel communication, I/O are taken care of**
- **Mesh data structures can be changed with minimal impact to application**
- **Speeds up application development process, easier learning curve**
- **Application code is more maintainable and extensible**
- **Particularly useful for mesh modifications and for parallel applications**

Representing a Mesh

- **Meshes are naturally related to the boundary representation (B-Rep) method for describing geometric models**
- **Can use a restricted B-rep method for describing meshes**
- **Regions $\rightarrow$ Faces $\rightarrow$ Edges $\rightarrow$ Vertices**
- **Meshes are naturally non-manifold – internal mesh faces are shared by two mesh regions**
- **No mesh entity can have a hole**
- **Typically, the geometry of meshes entities is also limited[1] (linear, quadratic, cubic)**

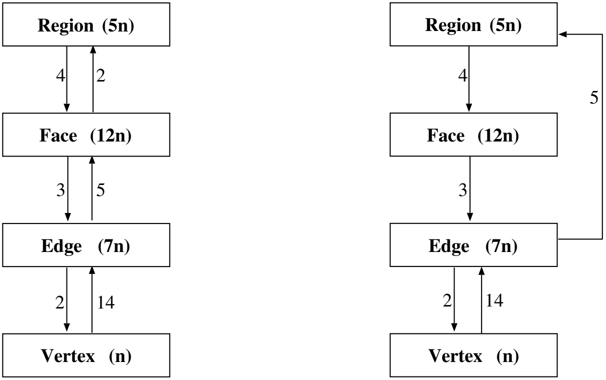
[1] except for isogeometric elements

Some Definitions for discussing Mesh Frameworks

- ***Topology***: Shape independent description of a mesh
- ***Geometry***: Shape of objects in a mesh
- ***Entity***: Any topological object in the mesh
- ***Region***: Object of topological dimension 3 (has volume)
- ***Face***: Object of topological dimension 2 (has area)
- ***Edge***: Object of topological dimension 1 (has length)
- ***Vertex (Node)***: Object of topological dimension 0 (occupies a point)
- ***Element***: Highest dimension entity in a mesh
- ***Adjacency***: Connectivity between any two types of mesh entities
- ***Mesh Representation***: Specific combination of entities and adjacencies in a mesh that are explicitly represented

Full Mesh Representations

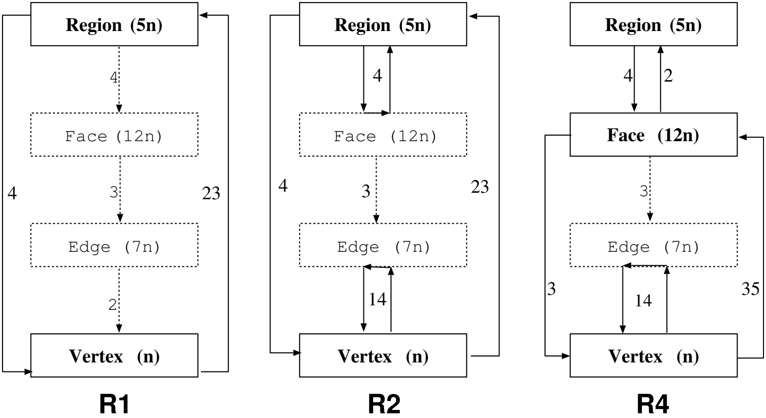
Mesh entities of all types are explicitly represented



F1 **F4**

Reduced Mesh Representations

Highest dimension entities (elements) and lowest dimension entities (nodes/vertices) are explicitly represented.



Intermediate entities are implicit.

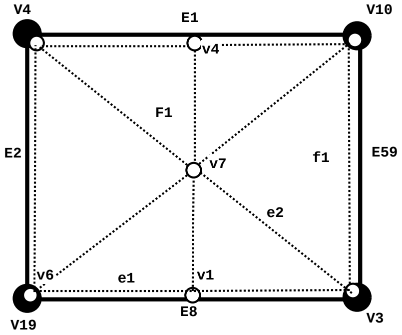
Which Representation Should I Use?

- **Reduced representations:**
  - **Compact but more expensive if intermediate entities are accessed**
  - **Use only if memory is scarce or intermediate entities are never needed**
  - **Need representations with faces for polyhedra (R4 or Fn)**
- **Full representations:**
  - **More efficient but use more memory**
  - **Required for meshes of non-manifold models with solids and free surfaces**
  - **Required if DOF live on intermediate entities**
- **Mixed representations - Full on the boundary only**
  - **More challenging to implement and manage**

Geometric Models and Meshes

- **Every mesh is a discrete representation of a continuous domain**
- **The domain may exist only as a conceptual model or may exist as a concrete model in a geometric modeling system**
- **Mesh classification is the relationship of a mesh to the geometric model it discretizes**
- **A mesh entity M is said to be classified on a geometric model entity G if M discretizes all or part of G but not its boundary**

Mesh Classification Example




```
v6 --> V19 (GEntDim=0, GEntID=19)
v1 --> E8  (GEntDim=1, GEntID=8)
v7 --> F1  (GEntDim=2, GEntID=1)

e1 --> E8  (GEntDim=1, GEntID=8)
e2 --> F1  (GEntDim=2, GEntID=1)

f1 --> F1  (GEntDim=2, GEntID=1)


Lower case letters: mesh entities
Upper case letters: geometric model
                    entities
```

Mesh Classification (Contd.)

How much classification information is required? As much as is available

- **Topological dimension of geometric model entity**
  - useful for constraints and boundary conditions
  - useful in avoiding dimensional reduction in mesh modifications
- **ID of geometric model entity**
  - useful for distinguishing material regions or surfaces
  - useful in enforcing topological conformity with geometric model
- **Handle to entity in geometric modeler**
  - useful for extracting topological and geometric details of domain
  - critical in adaptive mesh refinement

Acquiring Mesh Classification

- **Best to get mesh classification information from mesh generator**
- **Most mesh generators throw away classification info before exporting mesh**
- **Most mesh formats do not directly support transmission of classification information**
- **Mesh generators could transmit classification information through mesh based data**

Deriving Mesh Classification

- **All is not lost if mesh does not come with detailed classification info**
- **Detailed classification information can be inferred from partial data**
- **Minimum information required is the classification of elements (regions in 3D, faces in 2D)**
- **Derived classification might be inaccurate in some tricky situations**

Algorithm for Deriving Classification for 3D Meshes

- **Most meshes know which "material region" each mesh region belongs to (default is 1)**
- **Mesh faces connected to one mesh region**
  - **classified on a model face**
- **Mesh faces connected to two mesh regions are classified on a**
  - **model region if both mesh regions are classified on the same model region**
  - **model face if the two mesh regions are classified on different model regions**

Algorithm for Deriving Classification (Contd.)

- **Mesh edges connected to one mesh face**
  - **classified on a model edge**
- **Mesh edges connected to multiple mesh faces are classified on**
  - **on a model region, if edge is not connected to any mesh faces classified on a model face**
  - **a model face, if edge is connected to exactly two mesh faces classified on the same model face**
  - **a model edge, if edge is connected to two or more mesh faces classified on different model faces**
- **Similarly for vertices**

Geometric Subgrouping of Mesh Faces on Model Face

Subgroup boundary mesh faces based on dihedral angles

1. **Start with a list containing a mesh face classified on a model face F_1**
2. **Process the next mesh face of the list**
3. **Get the neighboring faces of the face (edge-connected neighbors only)**
4. **Put a neighboring face onto the queue if the two faces form a shallow angle**

   – dihedral angle between the face and its neighbor is above a threshold (135°)
5. **Repeat 2–4 until all faces in the list have been processed**
6. **The mesh faces of the list mark out a new model face that is "smooth" by our dihedral angle criterion**
   - **This new model face is a subset of F_1**
   - **Do not create new faces if all faces of F_1 are in the list**

Data Structures for Mesh Frameworks - Entity Level

- **structures or classes (recommended):**
  - **all the data for the entity is contained in one place**
  - **Addition or deletion of an entity is a single operation**

- **multiple arrays to store entity information:**
  - **e.g., one array each for element type, number of nodes of element, model entity ID of element**
  - **more messy and error-prone**
  - **addition or deletion of an entity requires access and updating of mutiple arrays**

Data Structures for Mesh Frameworks - Mesh Level

- **Linked lists: convenient for inserting/deleting entries**
- **Unfortunately, linked lists are memory hogs and are horribly inefficient for random access operations**

- **“Smart” arrays (or C++ standard vectors) are much more efficient containers**
- **Arrays offer constant time access of any element in a static mesh**
- **“Smart” arrays keep track of the number of entries and maximum alllocation**
- **They automatically expand when elements are appended**
- **They can be told to compress themselves**
- **Very little overhead over using arrays directly**

Data Structures for Dynamic Meshes

- **Some extra work with “smart” arrays for dynamic meshes**
- **Allow the array to have holes in case of entity deletions and add new entries only at the end**
- **If application requests element ’i’ in a dynamic mesh, the operator has to walk through part of the array stepping over holes**
- **However, the algorithm can be sure that element ’i’ will not be before the ’i’th location in the array**
- **Can compress the array occasionally to restore efficiency**
- **Compression of mesh arrays must be done carefully**
  - **All mesh related arrays must be updated**
  - **Code should not be iterating through mesh or mesh-based data**

Handling Mesh-based Application Data

- **Need to store application data tied to mesh entities**
- **Density (on elements), fluxes (on faces), velocities (on vertices), etc.**
- **Can store the data with the entity class or structure**

  – conceptually elegant but inefficient
- **Store the data in arrays mirroring entity IDs**
- **More efficient and in line with black box solvers and other computational tools**
- **May also have advantages w.r.t. memory access**
- **But ... have to account for deleted entries in dynamic meshes**
- **Handle gaps in data by compressing the mesh and data or copying to contiguous array**

Handling Mesh Modifications

- **Mesh modifications necessary for adaptivity, mesh quality improvement, mesh motion**
- **Node movement is trivial - topology change is more challenging**
- **Lowest set of mesh modifications are:**
  - **entity deletion – delete an edge**
  - **entity creation – add a vertex**
  - **entity modification – replace an edge vertex with another**
- **Low level mesh modifications must be used with care as they can mess up mesh topology badly**
- **Still, some safeguards can be built-in even for low level modifications**
- **For example, if an edge is deleted, code can internally tell the edge vertices that they must stop referencing this edge**

High Level Local Mesh Modifications

- **Edge/Face/Region split**
- **Edge collapse**
- **Edge/Face Swap**
- **Merge faces/edges with common boundaries**
- **Join faces/edges to form single entity**
- **Such operations take the mesh from one valid state to another**
- **Internal states may be invalid but calling application always gets back valid topology**
- **Useful to provide in the mesh framework library**

Role of Classification in Mesh Modification

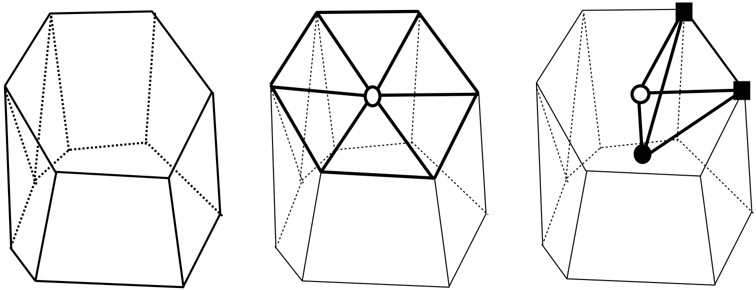
- **Classification information of a mesh can be used to guide mesh modification decisions**
- **Using just model entity dimension and ID of mesh entities:**
  - **Mesh/Model Topological conformity can be maintained**
  - **Possibility of dimensional reduction can be minimized**
- **Using the geometry of model entities in a geometric modeler,**
  - **Refinement points can be placed more accurately on surfaces**
  - **Boundary properties such as curvature can be evaluated to drive adaptation**

Checking Mesh Validity

- **Must have sanity checks to ensure mesh is valid**
- **Useful tool for checking imported and modified meshes**
- **Can check mesh topology, mesh geometry and conformity with geometric model**
- **Topological checks ensure that upward and downward adjacency information is consistent among entities**
  - **If an edge references a vertex, the vertex knows about the edge**
  - **Direction in which a region uses a face is consistent with which side of the face the region is on**
- **Conformity with geometric model**
  - **vertex on model edge is connected to two edges classified on the same model edge**
  - **vertex on model face has a ring of mesh faces classified on the same model face**

Geometric Validity of Elements

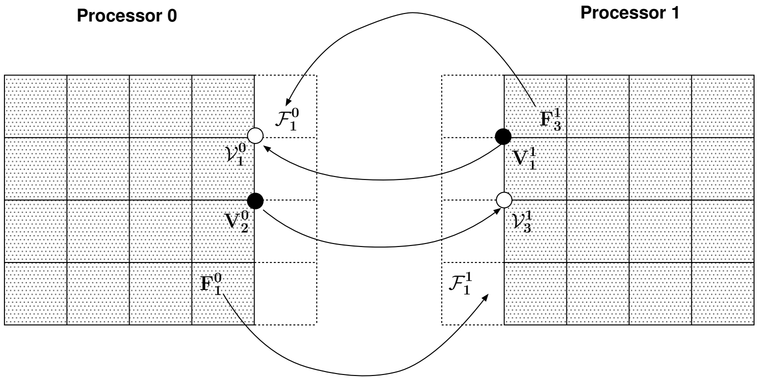
- Validity of tetrahedra is trivial - check if the signed volume of the tetrahedron is greater than zero
- Validity of more general elements such as hexahedra and general polyhedra is more complex
- Depends heavily on the application e.g. FE codes require that Jacobian be positive at quadrature points
- Can use star-shape test to capture many of these criteria for general shapes



Mesh Representation for Simple Parallel Architecture

- **Assume one processor per node and distributed memory**
- **Communication through MPI (Message Passing Interface)**
- **Typical mesh representation:**
  - **Core set of elements owned by the processor (Owned elements)**
  - **One or more layers of elements that are copies of elements owned by other processors (ghost elements)**
  - **Entities on interprocessor boundaries shared by multiple processors**
  - **Shared entities obey the Master-Slave paradigm**
  - **Slave entities are passive copies of master entities**
- **Ghost entities allow many computations at processor boundaries without interprocessor communication**
- **Parallel synchronization step only if data on owned or master entities changes**

Parallel Mesh Representation



- $\mathcal{F}_1^0$ **on processor 0 is the ghost of** F_3^1 **on processor 1**
- $\mathcal{F}_1^1$ **on processor 1 is the ghost of** F_1^0 **on processor 1**
- $\mathcal{V}_1^0$ **on processor 0 is the slave of** V_1^1 **on processor 1**
- $\mathcal{V}_3^1$ **on processor 1 is the ghost of** V_2^0 **on processor 1**

Parallel Meshes on Emerging Architectures

- **Emerging architectures going towards multi-layer model with a combined distributed+shared memory model**
- **Large numbers of “compute nodes” with small amounts of distributed memory**
- **Each node has multiple processors/cores sharing memory**
- **Many newer machines also include bank of GPUs for fast vector processing**

Parallel Meshes on Emerging Architectures

- **Proposed strategy: Overpartition the mesh and put multiple partitions on each compute node**
- **Each core on a compute node gets one partition**
- **Compute nodes exchange information between each other using traditional MPI calls**
- **Threading used for multiple processors on one node**
- **Must distinguish between partition boundaries that need MPI communication (inter-node boundaries?) and those that do not (inter-core boundaries?)**
- **May need to use reduced representations if memory on each node gets too small**
- **It is unclear (at least to me) how mesh frameworks and mesh related operations can make effective use of GPUs**

Other Practicalities of Handling Parallel Meshes

- **Partitioners:**
  - **Zoltan, Metis, ParMetis**
- **Parallel Communication:**
  - **MPI (distributed memory)**
  - **pthreads, OpenMP (shared memory)**
- **Parallel mesh file formats:**
  - **HDF5, Parallel netCDF, Nemesis (extension of Exodus II), CGNS (built on top of ADF/HDF5)**
- **Parallel I/O (still evolving for meshes)**
  - **Processor 0 reads and distributes (too slow for large meshes)**
  - **Every processor reads its own file (I/O chokes for too many processors)**
  - **Happy medium: subset of processors do I/O and distribute mesh to processors under their responsibility**

Mesh Framework and Related Software

Alphabetical list of full featured parallel mesh framework libraries. This list does not claim to be complete.

- **FMDB (Flexible Mesh Database), Rensselaer Polytechnic Institute**
- **MOAB (Mesh Oriented Data Base), Argonne National Lab**
- **MSTK (MeSh ToolKit), Los Alamos National Lab**
- **STKmesh (Sierra Toolkit Mesh), Sandia National Labs**

Other related software:

- **ITAPS (Interoperable Technologies for Advanced Petascale Simulations): Not a mesh framework but an interface specification**
- **GRUMMP (UBC) and NWgrid (PNNL) can be used as mesh framework libraries through ITAPS**
- **OpenMesh, OpenFoam, LaGriT, VTK, OpenFVM, LibMesh, Sieve, CACTUS, etc.**

MSTK

https://software.lanl.gov/MeshTools/trac

- **Supports multiple mesh representations (full and reduced)**
- **All linear element types including polygons and polyhedra**
- **Parallel meshes with one layer of ghosts**
- **Mesh entity attributes/data with parallel update**
- **Mesh modification but not in parallel**
- **C interface for developers**
- **Reads: MSTK, Exodus II, GMV formats**
- **Parallel I/O: Read mesh on one processor, partition and distribute**
- **Open source (LGPL)**

MOAB

http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB

- **Reduced mesh representations with option to create intermediate entities (faces, edges)**
- **All linear elements including polygons and polyhedra (?)**
- **Has structured mesh representation**
- **Parallel meshes with one or more layers of ghosts**
- **Mesh entity attributes/data with parallel update**
- **Mesh modification but not well suited for it (?)**
- **C/C++ interface for developers**
- **Reads: HDF5, Exodus II**
- **Parallel I/O: Read pre-partitioned mesh on each processor**
- **Open Source (LGPL)**

STKmesh

http://trilinos.sandia.gov/packages/stk/

- **Mesh representation with the option to create intermediate entities and adjacencies**
- **Linear and quadratic (?) elements including polygons and polyhedra (?)**
- **Parallel meshes with one layer of ghosts**
- **Mesh entity attributes/data with parallel update**
- **Mesh modification in serial and parallel**
- **Quite low level, one needs to custom build a “mesh class” from STKmesh classes**
- **Steep learning curve; Doxygen documentation but little else**
- **I/O: Exodus II**
- **Parallel I/O: Read pre-partitioned mesh on each processor (?)**
- **Open-source (BSD)**

FMDB

http://www.scorec.rpi.edu/FMDB/

- **Mixed mesh representation - full on the boundaries and reduced on the interior?**
- **Linear and higher order elements but not polygons/polyhedra (?)**
- **Parallel meshes with one or more layers of ghosts**
- **Mesh entity attributes/data with parallel update**
- **Mesh modifications in serial and parallel**
- **Reads: SMS, VTK, NetCDF**
- **Parallel I/O: Read pre-partitioned mesh on each processor**
- **Free for non-commercial use**

ITAPS

http://www.itaps-scidac.org/

- **Interoperable Technologies for Petascale Simulations**
- **DOE SciDac sponsored effort**
- **Not a mesh framework**
- **Common interface specification for handling meshes, geometry and field data in parallel applications**
- **Good way for different codes to interface together**
- **Need a concrete mesh framework library underneath to do the work**
- **MOAB, FMDB, GRUMMP and NWGrid have ITAPS interface implementations**
- **iMesh (mesh API in serial) and iGeom (geometry interface) fairly well hashed out**
- **iMeshP (parallel mesh), iField (mesh based data) are newer**

Setup of Pseudo-Shock Propagation Problem

- **Simulate a “shock” traveling across a distributed mesh**
- **“Shock” is represented by higher densities (10.0) at or behind than ahead (1.0)**
- **Start with a face in one corner with a high density**
- **At each “time step”, any low density face adjacent to a high density face is updated to have a high density**
- **At the end of each “time step”, parallel synchronization of data is performed**

“Shock” Propagation Code using MSTK

```
MSTK_Init();
MPI_Comm_size(MPI_COMM_WORLD,&num);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
.
.
.
mymesh = MESH_New(UNKNOWN_REP);

if (rank == 0) {
  MESH_InitFromFile(mymesh, filename);

  if (MESH_Num_Regions(mymesh) > 0) {
    fprintf(stderr,"Code is for surface meshes only. Exiting...\n");
  }
  else if (MESH_Num_Faces(mymesh) > 0)
    dim = 2;

  MSTK_Mesh_Distribute(&mymesh, dim, 1, 1, rank, num, MPI_COMM_WORLD);
}
```

“Shock” Propagation Code with MSTK (2)

```
/* Simulate a diagonal shock front moving across the mesh */

rhoatt = MAttrib_New(mymesh,"density",DOUBLE,MFACE);

idx = 0;
while ((mf = MESH_Next_Face(mymesh,&idx)))
  MEnt_Set_AttVal(mf,rhoatt,0,RHOMIN,NULL);


mfaces = List_New(0);

/* Find the cell(s) connected to the upper left corner vertex */

idx = 0;
while ((mv = MESH_Next_Vertex(mymesh,&idx))) {
  MV_Coords(mv,vxyz);

  if (vxyz[0] == 0.0 && vxyz[1] == 1.0) {
    vfaces = MV_Faces(mv);
    idx1 = 0;
    while ((vf = List_Next_Entry(vfaces,&idx1)))
      MEnt_Set_AttVal(vf,rhoatt,0,RHOMAX,NULL);
    List_Delete(vfaces);

    break;
  }
}
```

"Shock" Propagation Code with MSTK (3)

```
/* Synchronize across processors to update ghost data */

MSTK_UpdateAttr(mymesh, rank, num, MPI_COMM_WORLD);

/* Propagate the density value through the mesh */

nsteps = 10;
for (i = 0; i < nsteps; i++) {

  idx = 0;
  while ((mf = MESH_Next_Face(mymesh,&idx))) {
    if (MEnt_PType(mf) == PGHOST) continue;

    ffaces = get_surrounding_faces(mf); /* not an MSTK function */

    idx1 = 0; maxrho = RHOMIN;
    while ((ff = List_Next_Entry(ffaces,&idx1))) {
      MEnt_Get_AttVal(ff,rhoatt,&ival,&rval,&pval);
      if (rval > maxrho)
        maxrho = rval;
    }

    if (maxrho > RHOMIN)
      List_ChknAdd(mfaces,mf);
    List_Delete(ffaces);

  } /* while (mf = MESH_Next_Face.... */
```

"Shock" Propagation Code with MSTK (4)

```
  /* Now we have all the low density faces adjacent to
  high density faces - set them to have a high density */

  idx = 0;
  while ((mf = List_Next_Entry(mfaces,&idx)))
      MEnt_Set_AttVal(mf,rhoatt,0,RHOMAX,NULL);

  /* Synchronize across processor; Update data on ghosts */

  MSTK_UpdateAttr(mymesh, rank, num, MPI_COMM_WORLD);

  /* Write mesh file out with data to GMV file */

  sprintf(gmvfilename,"%s.gmv.%04d.%04d",basename,rank,i+1);
  MESH_ExportToGMV(mymesh,gmvfilename,0,NULL,NULL);

} /* for (i = 0; i < nsteps; i++) */

MPI_Finalize();
```

How to Use and Evaluate Frameworks

- **Build an additional layer between application and mesh framework**
  - **Use ITAPS if it meets all your requirements**
  - **Use your own lightweight wrappers**
- **Extra layer of protection against bugs, deficiencies in the frameworks**
- **Code can leap-frog over roadblocks - if one framework fails, continue development with another**
- **Allows you to compare apples-to-apples**
- **Other developers deal with a simpler, application-relevant roster of mesh queries**
- **Can build in application-specific caching for efficiency**
- **If the extra layer truly slows you down, you can easily eliminate it later**

Example of Framework-Neutral Interface

- **ASCEM is a large DOE effort to develop an community code for subsurface flow and transport (http://ascemdoe.org)**
- **HPC code Amanzi uses a framework neutral mesh interface that looks like this:**
  - int num_entities(...)
  - void cell_get_faces(...)
  - void cell_get_face_dirs(...)
  - void node_get_cells(...)
  - Epetra_Map cell_epetra_map(...)
  - bool valid_set_name(...)
- **Exposes 30 mesh queries instead of the 100-200 calls in each framework**
- **1000 processor runs of flow and transport computations on real geometries with more targeted this year**

Geometric Models in Meshing and Simulation

- **Need access to underlying geometric model for**
  - **Surface mesh generation**
  - **Mesh adaptation**
  - **Higher order element creation**
  - **Application of variable boundary conditions**
  - **Isogeometric shape function evaluation**

- **Typically need to query**
  - **topology**
  - **normals and curvatures**
  - **closest points**
  - **parametric to real coordinate map**
  - **line-surface intersection, etc.**

Geometric Models in Meshing and Simulation (2)

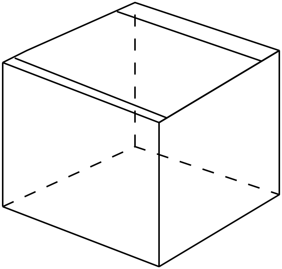
- **Different users may use different geometric modelers**
- **Commonly used geometric modeling kernels are Parasolid, ACIS, Granite and Open Cascade**
- **Many similarities**
  - **B-rep type queries of topology**
  - **analytical, B-Splines, Beziers, NURBS geometry**
- **Some differences:**
  - **Support for non-manifold models**
  - **Tolerant models**

Geometric Modeler Interfaces

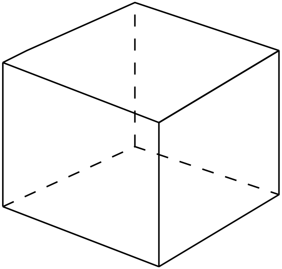
- **Applications should interact with geometric modelers through a functional interface**
- **Enables the same code to work with multiple geometric modelers**
- **Consider more than a simple wrapper around the modeler**
- **Such a layer can support creation of an altered view of the model modeler interface**

Geometric Model Alteration in Interface Layer (1)

Can be used to "stitch" two faces with an unecessary seam or hide a small edge or face from the meshing algorithm



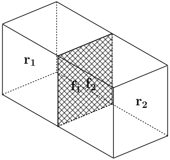
Model in CAD system



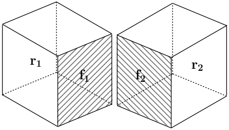
Simpler view of model in interface layer

Geometric Model Alteration in Interface Layer (2)

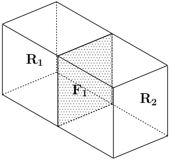
Also can be used to create a non-manifold view of the model if the geometric modeler does not support it



Model in CAD system
Two Disjoint Regions
with coincident faces



Exploded view showing the two solids
f_1 knows only about r_1
f_2 knows only about r_2



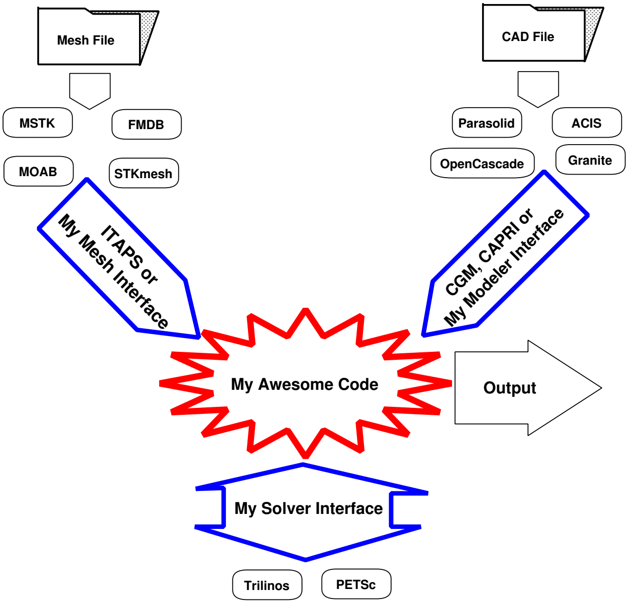
Model in interface layer

F_1 knows about both regions R_1 and R_2

Available Geometric Modeler Interfaces

- **CAPRI or Computational Analysis PRogramming Interface**
  - **http://raphael.mit.edu/mmodel.pdf**
  - **Commercially available through CADNexus Inc.**
- **CGMA or Common Geometry Module/Argonne:**
  - **http://trac.mcs.anl.gov/projects/ITAPS/wiki/CGM**
  - **Open Source**
  - **ACIS, Open Cascade, CUBIT**
- **iGeom: Not a library but a specification developed for ITAPS**

Final Picture



Closing Remarks

- **If you are developing a large scale computational code using unstructured meshes, use a mesh framework**
- **If your code needs to go parallel, definitely use a mesh framework**
- **Talk to the framework developers even if you think your needs are unusual and you need to write your own**
- **Access geometric models through a geometric modeler interface library**

Closing Remarks (contd.)

- **Mesh frameworks saves you development time, debugging time and a lot of heartburn**

- **Any (minor) drop in efficiency is offset in development time, and maintanability and extensibility of the code**

  Besides, who knows if that efficiency gain in accessing arrays directly will persist after the data structures have been repeatedly band-aided over the years?

Object Oriented Paradigm

- **Meshes and mesh entities are perfect for C++ - you can easily think of many mesh constructs as "it"**
- **If you code in C++, great!**
- **If you don't, still consider the object oriented paradigm**
- **Treat meshes and mesh entities as objects with**
  - **Private data**
  - **Well defined public and private methods**
- **Harder to do in Fortran than in C but not impossible**
- **Of course, unlike C++, programmers can circumvent object-orientation in C and Fortran**
- **Still, such a code offers many of the benefits of the OO methodology**

Future directions

- **Emerging architectures will pose new challenges for working with unstructured meshes**
- **We may all have to use a combination of MPI, threads and GPU programming?**